Что такое технология Java?

**Ответ.** Технология Java - это одновременно язык программирования и платформа.

Язык программирования Java является объектно-ориентированным языком высокого уровня.

Платформа - это аппаратно-программная среда, в которой выполняется java-программа. Платформа Java состоит из двух компонентов:

1. виртуальная машина Java,

2. Java API.

**Источник.** <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>

Что такое JVM?

**Ответ.** Java Virtual Machine (сокращенно Java VM, JVM) - виртуальная машина Java - основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE). Виртуальная машина Java интерпретирует Байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac).

**Источник.** <https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-1.html#jvms-1.2>

Что такое JDK?

**Ответ.** Java Development Kit (сокращенно JDK) - комплект разработчика приложений на языке Java, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java (JRE).

**Источник.** <https://docs.oracle.com/javase/8/docs/index.html>

Что такое JRE?

**Ответ.** Java Runtime Environment (сокр. JRE) - минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений, без компилятора и других средств разработки. Состоит из виртуальной машины - Java Virtual Machine и библиотеки Java-классов.

**Источник.** [**https://docs.oracle.com/javase/8/docs/index.html**](https://docs.oracle.com/javase/8/docs/index.html)

Даны корректные исходные файлы: Hello.java и hello.cpp. Каждый из них преобразуется в исполняемый код, который запускается на выполнение. Назовем данный процесс жизненным циклом приложения. В чем отличие жизненных циклов этих приложений?

**Ответ.** Исходная Java-программа Hello.java транслируется в байт-код компилятором javac.exe. Оттранслированная в байт-код программа имеет расширение class . Для запуска программы нужно вызвать интерпретатор java.exe , указав в параметрах вызова, какую программу ему следует выполнять. Кроме того, ему нужно указать, какие библиотеки нужно использовать при выполнении программы. Библиотеки размещены в файлах с расширением jar.



Рисунок 1. Жизненный цикл приложения на Java

Для того чтобы выполнить исходную программу hello.cpp, требуется перевести ее на язык, понятный процессору — в машинные коды. Этот процесс состоит из нескольких этапов.

Сначала программа передается препроцессору, который выполняет директивы, содержащиеся в ее тексте.

Получившийся полный текст программы поступает на вход компилятора, который строит объектный модуль.

Компоновщик формирует исполняемый модуль программы, подключая к объектному модулю другие объектные модули, в том числе содержащие функции библиотек, обращение к которым содержится в любой программе. Если программа состоит из нескольких исходных файлов, они компилируются по отдельности и объединяются на этапе компоновки. Исполняемый модуль имеет расширение ехе.



Рисунок 2. Жизненный цикл приложения на C++

Отличия:

1. Приложения java запускаются в виртуальной среде JVM, а C++ в операционной системе.
2. Сlass-файлы Java в отличие от приложений С++ не содержат инструкций для конкретного процессора и могут запускаться на разных платформах.
3. Программы java не могут напрямую работать с памятью в отличие от С++
4. Все стандартные библиотеки уже включены в JRE их не нужно добавлять в исходные файлы на Java

**Источник.**<http://www.javable.com/tutorials/fesunov/lesson1/>,<http://itandlife.ru/programming/cpp/etapy-kompilyacii-i-komponovki-programm-na-yazyke-c/>

<http://itandlife.ru/programming/cpp/etapy-kompilyacii-i-komponovki-programm-na-yazyke-c/>

<http://www.realcoding.net/teach/java/Glava0/Index3.htm>

Перечислите группы примитивных типов данных и состав каждой группы.

**Ответ.** В Java есть 8 примитивных типов данных:

1. Числовые. Делятся на:

a. Целочисленные - byte, short, int, long, char

b. С плавающей точкой - float, double

2. Логический - boolean

**Источник.** <https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2>

Какой размер данного булевского типа?

**Ответ**. Размер данного булевского типа не в массивах - 32 бита, так как используется тип int, и 8 бит в массивах, так как там используется тип byte.

**Источник.** <https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.3.4>

Какой размер данного символьного типа?

**Ответ.** Размер символьного типа данных составляет 16 бит (положительное целое число от 0 до 65535 включительно, представляющее символ в кодировке UTF-16).

**Источник.** <https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.2>

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Какими литералами можно инициализировать символьную переменную?

**Ответ:**

1. символьный литерал, заданные напрямую любым отображаемым символом в одинарных кавычках, например ‘A’;

2. ‘\uxxxx’ - символ Unicode, гду xxxx цифровой код символа Unicode в шестнадцатеричной форме;

3. ‘\xxx’ - символ кодовой таблицы Unicode, где xxx - восьмеричный код символа;

4. Управляющие символы (‘\n’ - конец строки; ‘\t’ – табуляция; ‘\b’ - забой; ‘\f’ - конец страницы; ‘\r’ - возврат каретки; ‘\”’ - двойная кавычка; ‘\’’ - одинарная кавычка; ‘\\’ - обратный слэш);

5. целым числом, не превышающим 16бит (в двоичной, восьмеричной, десятичной, или шестнадцатеричной форме).

**Источник:**<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Какие есть виды переменных?

**Ответ.**

1. Переменная класса – со спецификатором static в объявлении класса или с/без спецификатора в интерфейсе.

2. Переменная экземпляра – без спецификатора static в объявлении класса.

3. Компоненты массива - это неназванные переменные, которые создаются и инициализируются значениями по умолчанию всякий раз, когда создается новый объект, являющийся массивом.

4. Параметры метода - значения аргументов, передаваемые методу.

5. Параметры конструктора - значения аргументов, передаваемые конструктору.

6. Лямбда-параметры - значения аргументов, передаваемые в тело лямбда-выражения.

7. Параметр исключения - создается каждый раз, когда исключение перехватывается блоком catch после блока  try.

8. Локальные переменные - объявляются с помощью операторов объявления локальных переменных.

**Источник.** <https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.12.3>

В какой области памяти хранятся локальные переменные?

**Ответ.** Локальные переменный хранятся в стеке (stack).

**Источник.**<https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.2>

В какой области памяти хранятся аргументы метода?

**Ответ.** Аргументы метода хранятся в стеке (stack).

**Источник.**<https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.2>

В какой области памяти хранятся объекты?

**Ответ.** В java все объекты хранятся в куче (heap).

**Источник.** <https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.3>

В какой области памяти хранятся ссылки на объекты?

**Ответ.** Cсылки на объекты хранятся в стеке (Java Virtual Machine stack), но если ссылка на объект является полем объекта, то данная ссылка вместе с объектом хранится в куче (heap) вместе с этим объектом.

**Источник.** <https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.3>

В какой области памяти хранятся массивы?

**Ответ**. Массивы хранятся в куче (heap).

**Источник**.[​https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.3](https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.3)

Дайте определение массива.

**Ответ**. Массив — это совокупность элементов:

1. одного и того же типа типа,
2. постоянная по размеру,
3. расположенных в памяти непосредственно друг за другом.

**Источник.**<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>

Назовите 3 этапа создания массива. Какие действия выполняет JVM на каждом этапе?

**Ответ.** 1. Объявление. (declaration) - JVM создает переменную типа ссылка (reference) на массив, содержащую тип массива.

2. Создание (instantiation) - JVM выделяет память под указанное количество элементов массива и для неинициализированного массива устанавливает значения по умолчанию для элементов. Ссылка связывается с массивом.

3. Инициализация массива. (initialization). JVM присваивает элементам массива конкретные значения (не по умолчанию).

**Источник.**<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>

<http://pr0java.blogspot.com/2015/05/1.html>

Приведите различные способы инициализации массива.

**Ответ.** Способы инициализации массива:

1. на этапе объявления:

А. int[] ar = {1,2}.

Б. int[] ar = new int[]{1,2}.

2. на этапе создания:

А. int[] ar;

ar = new int[]{1,2}.

Б. Присваивание значений элементам массива с использованием индексов (для этого можно использовать циклы):

String[] seasons = new String[4];

seasons[0] = "Winter";

seasons[1] = "Spring";

seasons[2] = "Summer";

seasons[3] = "Autumn".

**Источник.** <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>

Дан массив

int[] values = {...};

//в фигурных скобках корректный список инициализации.

Есть ли различие в проходе по массиву циклами:

1. for(int i=0; i < values.length; i++) {...}

2. for(int value : values) {...}

Если да, то в чем различие заключается?

Как называется каждый цикл?

**Ответ.**

1. for(int i=0; i < values.length; i++) {...} - цикл с параметрами, цикл по счетчику, цикл по записи

2. for(int value : values) {...} - цикл foreach, цикл по чтению

Разница:

1. В for цикле при изменении индекса i в теле цикла или использовании прерывания можно изменить порядок прохода по массиву. В for each цикле проход по массиву происходит с помощью итератора полностью через все элементы по порядку.

2. В for цикле при проходе по массиву мы напрямую обращаемся к элементу массива и таким образом можем его изменить, а в for each – нет, т.к. в данном цикле мы работаем с итерационной переменной, а не с элементом массива.

3. Использование for each цикла не наносит никакого ущерба производительности. На самом деле такой цикл может предложить небольшое преимущество в производительности над циклом for в некоторых обстоятельствах, так как он подсчитывает границу индекса массива только один раз.

4. В for each цикле исключается возможность выхода за пределы массива.

**Источник:**<https://docs.oracle.com/javase/specs/jls/se8/html/jls-14.html#jls-14.14>

<http://pr0java.blogspot.com.by/2015/05/1.html>

Можно ли создать массив нулевой длины? Если да, то для каких целей он необходим?

**Ответ.** Можно создавать и использовать массивы нулевой длины (пустой массив). Например:

boolean[] bits = new boolean[0];

Инициализировать такой массив нельзя, так как у него просто нет элементов, которые можно инициализировать.

Пустой массив принято использовать в тех местах программы, где заранее неизвестно, будут элементы или нет.

Плюсы:

1. Он избегает необходимости проверять значение null, если предварительным условием является то, что массив всегда присутствует.
2. Можно использовать методы, применимые к массивам и перебирать массивы нулевой длины (чего нельзя делать с null)

**Источник.** <http://pr0java.blogspot.com/2015/05/1.html>

Что собой представляет двумерный массив? Многомерный массив?

**Ответ.** Двумерный массив в Java — это массив массивов, то есть в каждой его ячейке находится ссылка на некий массив. Но гораздо проще его представить в виде таблицы, у которой задано количество строк (первое измерение) и количество столбцов (второе измерение). Двумерный массив, у которого все строки имеют равное количество элементов, называется прямоугольным.

В обобщенном виде многомерные массивы в Java выглядят так:

DataType[dimension1][dimension2][]..[dimensionN] arrayName = new DataType[size1][size2]….[sizeN];

Где DataType — это тип элементов в массиве. Может быть примитивным или ссылочным (классом).

Количество пар скобок с dimension внутри — размерность массива (в нашем случае — N).

arrayName — название массива

size1...sizeN — количество элементов в каждом из измерений массива.

**Источник.** <https://javarush.ru/groups/posts/mnogomernye-massivy>

Перечислите ссылочные типы.

**Ответ.** **Ответ.** Существует четыре типа ссылочных данных в Java:

1. Классы (class types)

1а. Перечисления (enums)

2. Интерфейсы (interface types)

3. Переменные типов (type variables)

4. Массивы (array types)

**Источник.** <https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.3>

Какая структура у java-файла (внутренности класса не указывать)?

Какие из элементов структуры являются обязательными?

**Ответ.** Общая форма файла, содержащего исходный код Java, может быть следующая:

- одиночный оператор package (необязателен);

- любое количество операторов import (необязательны);

- одиночный открытый (public) класс (необязателен);

- любое количество классов пакетного уровня доступа (необязательны);

- описания интерфейсов (необязательны);

- комментарии (необязательны).

Классы в файле могут отсутствовать. При этом в результате компиляции не будет создан ни один файл с расширением .class.

**Источник.** И.Н. Блинов, В. С. Романчик, Java. Методы программирования, стр. 120.

Для чего предназначены пакеты?

**Ответ.** Java package (пакет Java) – механизм, позволяющий организовать Java классы в пространстве имен. Пакеты предназначены для:

1. Задания пространства имен, предотвращения коллизий имен классов
2. Логической группировки связанных классов
3. Инкапсуляции - сокрытия деталей реализации за счет модификаторов доступа

**Источник.** <http://pr0java.blogspot.com/2015/06/java.html>

Как выглядит полное имя класса?

**Ответ.** 1. Полное имя класса верхнего уровня или интерфейса верхнего уровня, которое объявлено в неназванном пакете, является простым именем класса или интерфейса.

2. Полное имя класса верхнего уровня или интерфейса верхнего уровня, объявленного в именованном пакете, состоит из полного имени пакета, за которым следует «.», за которым следует простое имя класса или интерфейса.

3. Дочерний класс или интерфейс может иметь полное имя, если родительский класс имеет полное имя. В таком случае оно выглядит - полное имя родительского класса, за которым следует «.», за которым следует простое имя дочернего класса или интерфейса.

4. Локальный класс не имеет полностью определенного имени

**Источник.** <https://docs.oracle.com/javase/specs/jls/se8/html/jls-6.html#jls-6.7>

Для чего предназначен оператор import?

**Ответ**. Полные имена классов могут быть слишком большими и писать их каждый раз при обращении к какому-либо классу из другого пакета затруднительно. Оператор import позволяет избежать этого делая доступным обращение к классу только по его (короткому) имени.

**Источник**. <https://docs.oracle.com/javase/tutorial/java/package/usepkgs.html>

Для чего предназначен статический импорт?

**Ответ.** Статический импорт дает возможность импортировать статические поля и статические методы классов и интерфейсов. Благодаря этому появляется возможность ссылаться на них непосредственно по их именам, не уточняя их именем класса. Статический импорт должен использоваться очень экономно. Чрезмерный статический импорт может привести к тому, что код трудно читать и поддерживать, потому что читатели кода не знают, какой класс определяет определенный статический объект.

**Источник**. [​https://docs.oracle.com/javase/1.5.0/docs/guide/language/static-import.html](https://docs.oracle.com/javase/1.5.0/docs/guide/language/static-import.html)

<https://docs.oracle.com/javase/specs/jls/se8/html/jls-7.html#jls-7.5.3>

Сколько внешних публичных классов можно объявить в одном java-файле?

**Ответ.** На каждую единицу компиляции (файл \*.java) должно быть не более одного внешнего публичного класса и любое количество классов пакетного уровня доступа. Это ограничение облегчает компилятору Java поиск именованного класса в пакете.

**Источник.** <https://docs.oracle.com/javase/specs/jls/se8/html/jls-7.html#jls-7.6>

Может ли синтаксически правильный java-файл иметь имя 1.java? Если да, то какие ограничения?

**Ответ.** Да, может. Но только в том случае, если в файле нет внешнего public класса. После компиляции файла 1.java компилятор создаст файл или файлы с расширением .class по именам package-private классов. Если же внешний public класс есть, то имя файла должно совпадать с именем public класса, а имя класса не может начинаться с цифры.

**Источник.** <https://docs.oracle.com/javase/tutorial/java/package/createpkgs.html>

Допустим, в одном java-файле объявлены два синтаксически правильных внешних класса. Какие могут быть результаты компиляции данного файла?

**Ответ**. Возможны 2 варианта результата компиляции:

1. Ошибка компиляции:

1.1. В случае, если оба класса объявлены с модификатором доступа public;

1.2. либо имя public класса не совпадает (с учетом регистра) с именем файла, в котором он объявлен.

2. Успешная компиляция:

2.1. Если имя public класса совпадает (с учетом регистра) с именем файла, в котором он объявлен, а второй класс имеет пакетный уровень доступа (package private);

2.2. либо в файле объявлены 2 класса с пакетным уровнем доступа.

В результате успешной компиляции будут созданы 2 class-файла, имена которых будут совпадать с именами классов, объявленных в исходном файле.

**Источник**.<http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/javac.html>

Как записывается заголовок метода main() раннер-класса (т.е. класса, который может стартовать из-под JVM)?

Это единственный способ?

**Ответ.**

public static void main(String[] args)

Нет. Можно еще так:

public static void main(final String... args)

Вместо имени аргумента args можно ставить любое свое. Но это общепринятая форма. Суть в том, что метод-раннер должен быть публичным, статичным, с возвращаемым типом данных void, иметь имя main и принимать массив типа String (хоть нулевой длины). Поэтому данная форма записи также соответствует требованию и метод распознается как раннер.

**Источник.** <https://docs.oracle.com/javase/tutorial/getStarted/application/index.html>

<https://www.baeldung.com/java-main-method>

Может ли один java-файл содержать два и более раннер-класса?

**Ответ.** Может. Но только один такой раннер-класс может быть public, а остальные

package-private раннер-класы. Либо все такие раннер-классы должны быть с уровнем

доступа раннер-класс. Имена классов внутри пакета должны быть разными (иначе

ошибка компиляции).

public class Test {

public static void main(String[] args){

System.out.println("test");

}

}

class BigTest {

public static void main(String[] args) {

System.out.println("bigtest");

}

}

**Источник.**<https://docs.oracle.com/javase/tutorial/java/package/createpkgs.html>

<https://docs.oracle.com/javase/specs/jls/se8/html/jls-7.html#jls-7.6>

Можно ли определить метод main() с другими модификаторами?

**Ответ.** Да, данный метод можно определять с другими модификаторами. Однако для запуска программы необходим метод с модификаторами public static void, а единственным аргументом метода должен быть массив строкового типа.

**Источник.** Хабибуллин И. Ш., Java 7.

Может ли в классе быть два и более метода main()?

Если да, то приведите пример.

**Ответ.** Да. Благодаря перегрузке методов. Например:

1. public static void main(String[] args){} - именно он будет точкой входа в программу.

2. public static void main(int[] args){}

**Источник**.<https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html>

Может ли в классе быть два метода main(), один из которых раннер, а второй метод main() отличается от первого только отсутствием модификатора static?

Если да, то приведите пример вызова второго метода.

**Ответ.** Не может. Модификатор static не входит в сигнатуру метода.

Два метода main() в классе могут быть только в случае перегрузки, когда они имеют разные параметры.

public **static** void main**(String[] args)**

public void main**()**

**Источник**.<https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html>

В корне проекта имеется файл HelloWorld.java, пути к необходимым ресурсам открыты.

Создайте bat-файл для запуска приложения на выполнение.

**Ответ.**

javac HelloWorld.java

java HelloWorld

В корне проекта имеется

1. папка src для исходных java-файлов, а в ней – файл HelloArgs.java, который выводит приветствие для каждого аргумента командной строки;

2. папка bin для class-файлов.

Пути к компилятору и JVM открыты, переменная CLASSPATH не установлена.

Создайте bat-файл для запуска приложения с двумя аргументами командной строки clever и 25007 на выполнение.

**Ответ.**

javac -sourcepath src -d bin src/HelloArgs.java

java -classpath bin HelloArgs clever 25007

**Источник**.<http://skipy.ru/technics/likbez.html>

Имеется корректный файл helloWorld.jar.

Создайте bat-файл для запуска приложения на выполнение.

**Ответ**.

java -jar helloWorld.jar

**Источник**.<http://skipy.ru/technics/likbez.html>

Какие есть виды комментариев?

**Ответ.** 1. «Однострочные» - // от начала символов и до конца строки.

2. «Блочные (многострочные)» - /\* все что находится между данными символами комментарий \*/.

3. «Документирующие» - Этот вид комментариев служит для создания НТМL-файла документации на программу, документирующий комментарий начинается с символов / \*\* и оканчивается символами \*/.

**Источник.** <https://docs.oracle.com/javase/tutorial/getStarted/application/index.html>